Secure Software Design

Design Defects = Flaws

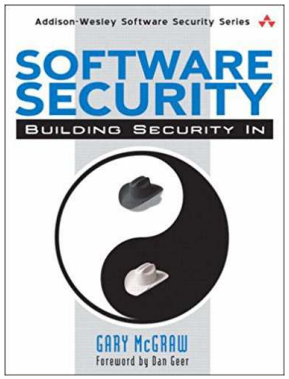
Recall that software defects consist of both flaws and bugs

- **Flaws** are problems in the **design**
- **Bugs** are problems in the **implementation** (e.g., buffer overruns)

The goal of the design phase from a security preservative is to avoid flaws during the design phase

According to Gary McGraw, **50% of security problems are flaws**

- So this phase is very important



Design vs. Implementation?

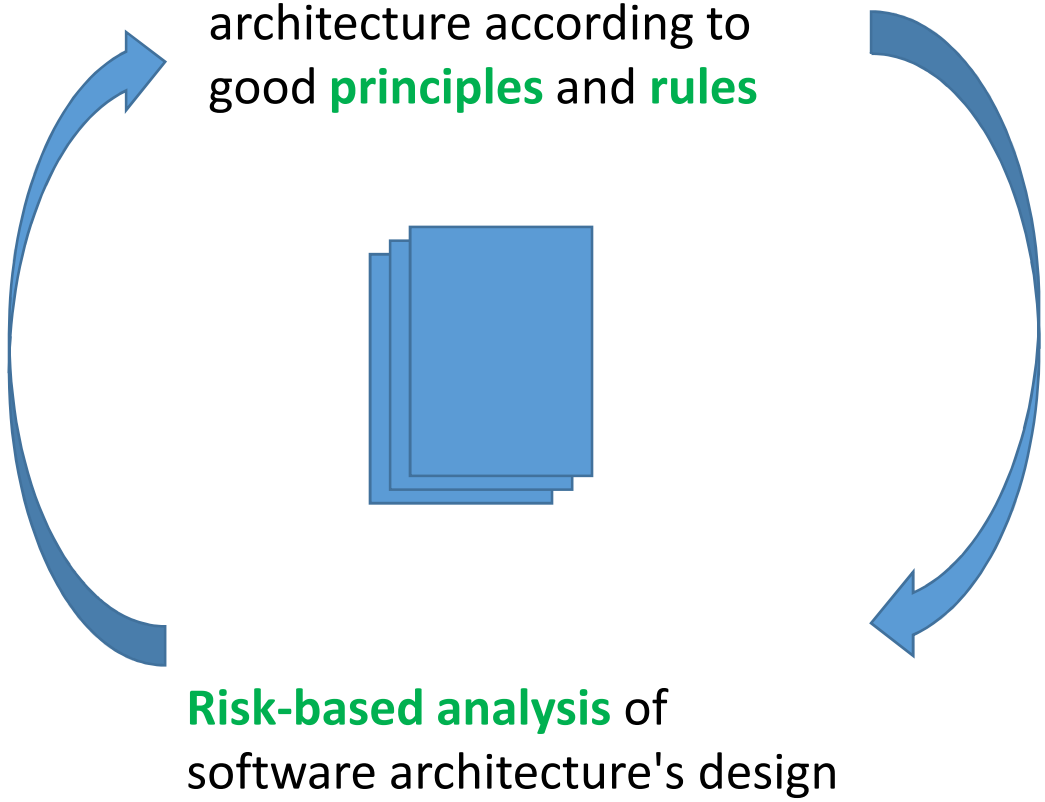
Many different levels of system design decisions

- Highest level: main actors (**processes**), **interactions**, and programming language(s) to use
- Next level: **decomposition** of an actor into **modules/ components**, identifying the core functionalities and how they work together
- lowest level: how to **implement data types** and **functions**, e.g., purely functionally, or using parallelism, etc.

Last two could be implementation or design, or both

- The distinction is a bit fuzzy

Secure Software Design



Principles and Rules

A **principle** is a high-level design goal with many possible manifestations

A **rule** is a specific practice that is consonant with sound design principles

- The difference between these two can be fuzzy, just as design vs. implementation is fuzzy.
  - For example, there is often a principle underlying specific practices

Principles often overlap

The **software design phase** tends to focus on **principles** for avoiding flaws

Categories of Principles

1. **Prevention**
   - **Goal**: Eliminate software defects entirely
   - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java
2. **Mitigation**
   - **Goal**: Reduce the harm from exploitation of unknown defects
   - **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another
3. **Detection** (and **Recovery**)
   - **Goal**: Identify and understand an attack (and undo damage)
   - **Example**: Monitoring (e.g., expected invariants), snapshotting

The Principles

1. **Favor simplicity**
   - Use fail-safe defaults
   - Do not expect expert users
2. **Trust with reluctance**
   - Employ a small trusted computing base
   - Grant the least privilege possible
     - Promote privacy
     - Compartmentalize
3. **Defend in Depth**
   - Use community resources - no security by obscurity
4. **Monitor and trace**

Classic Advice

The classic reference on principles of secure design is **The Protection of Information in Computer Systems**, by Saltzer and Schroeder (in 1975)

Principles

Economy of Mechanism

Fail-safe Defaults

Complete mediation

Open design

Psychological acceptability

- **Separation of privilege**
- **Least privilege**
- **Least common mechanism**
- **(Work factor)**
- **(Compromise recording)**

http://web.mit.edu/Saltzer/www/publications/protection/Basic.html

Comparing to our list

Several principles reorganized/renamed

- *Separation of privilege* has elements of our **compartmentalization**, **defend in depth**
- *Open design* is like **use community resources**, but did not anticipate open-source code

Monitoring is added

- Their focus on prevention of attack, rather than recovery

"Principle" of *complete mediation* dropped

- CM not a *design* principle, but a rather an implementation requirement

Design Category: Favor Simplicity

Favor Simplicity

Keep it **so simple** it is **obviously correct**

- Applies to the external interface, the internal design, and the implementation
  - Classically referred to as **economy of mechanism**
- **Category**: Prevention
  - avoiding defects in the first place

"We've seen **security bugs in almost everything**: operating systems, applications programs, network hardware and software, and security products themselves. **This is a direct result of the complexity of these systems.** The more complex a system is--the more options it has, the more functionality it has, the more interfaces it has, the more interactions it has—the harder it is to analyze [its security]". *—Bruce Schneier*

https://www.schneier.com/essays/archives/1999/11/a_plea_for_simplicit.html

FS: Use fail-safe defaults

Some **configuration** or **usage choices affect** a system's **security**

- The length of cryptographic keys
- The choice of a password
- Which inputs are deemed valid

The default choice should be a secure one

- **Default key length is secure** (e.g., 2048-bit RSA keys)
- **No default password**: cannot run the system without picking one
- **Whitelist** valid objects, rather than blacklist invalid ones
  - E.g., don't render images from unknown sources

Hackers Breach Security of HealthCare.gov

As an example of a failure to apply a fail safe default, consider the breach of healthcare.gov.

In this case it was possible because the server was connected to the internet with a default password still enabled.

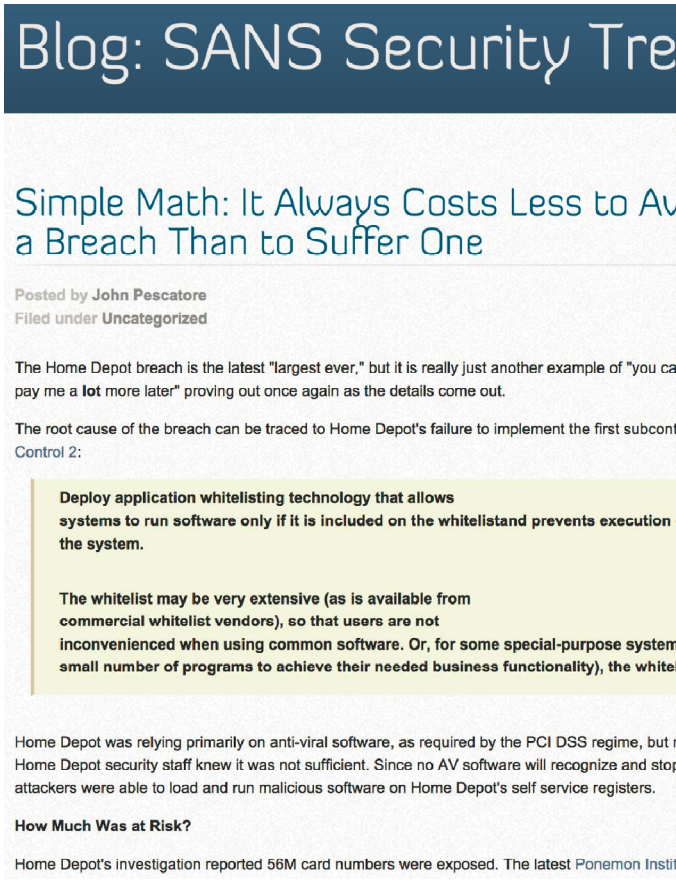
"….Mr. Albright said the hacking was made possible by several security weaknesses. The test server should not have been connected to the Internet, he said, and it came from the manufacturer with a default password that had not been changed."

ttps://www.nytimes.com/2014/09/05/us/hackers-breach-security-of-healthcaregov.html

The Home Depot breach

"*Deploy application whitelisting technology that allows systems* to run software only if it is included on the whitelist and prevents execution of all other software on the system"

Whitelisting on servers and single function servers or appliances has proven to cause near zero business or IT administration disruption



Blog: SANS Security Tre

Simple Math: It Always Costs Less to Av
a Breach Than to Suffer One

Posted by **John Pescatore**
Filed under **Uncategorized**

The Home Depot breach is the latest "largest ever," but it is really just another example of "you ca
pay me a **lot** more later" proving out once again as the details come out.

The root cause of the breach can be traced to Home Depot's failure to implement the first subcont
Control 2:

> **Deploy application whitelisting technology that allows systems to run software only if it is included on the whitelistand prevents execution the system.**
>
> **The whitelist may be very extensive (as is available from commercial whitelist vendors), so that users are not inconvenienced when using common software. Or, for some special-purpose system small number of programs to achieve their needed business functionality), the white**

Home Depot was relying primarily on anti-viral software, as required by the PCI DSS regime, but
Home Depot security staff knew it was not sufficient. Since no AV software will recognize and stop
attackers were able to load and run malicious software on Home Depot's self service registers.

How Much Was at Risk?

Home Depot's investigation reported 56M card numbers were exposed. The latest Ponemon Instit

://www.sans.org/security-trends/2014/09/23/simple-math-it-always-costs-less-to-avoid-a-breach-than-to-
r-one

FS: Do not expect expert users

Software designers should consider how the **mindset and abilities** of (the least sophisticated of) a system's **users** will **affect security**

Favor simple user interfaces

- **Natural or obvious choice is the secure choice**
  - Or avoid choices at all, if possible, when it comes to security
- **Don't have users make frequent security decisions**
  - Want to avoid user fatigue
- **Help users explore ramifications of choices**
  - E.g., allow admin to explore user view of set access control policy

Passwords

Goal: easy to remember but hard to guess

- Turns out to be **wrong** in many cases
  - **Hard to guess = Hard to remember!**
- **Compounding problem: repeated password use**

Password cracking tools train on released data to quickly guess common passwords

- John the Ripper, http://www.openwall.com/john/
- Project Rainbow, http://project-rainbowcrack.com/
- many more …

Top 10 worst passwords of 2016:123456, password, 12345, 12345678, football, qwerty, 1234567890, 1234567,princess, 1234, login, welcome [from SplashData]

- https://www.teamsid.com/worst-passwords-2016/

Password Manager

A password manager (PM) stores a **database of passwords, indexed by site**

- Encrypted by a **single, master password** chosen (and remembered) by the user, used as a key
- **PM generates complicated per-site passwords**
  - Hard to guess, hard to remember, but the latter doesn't matter

Benefits

- Only a single password for user to remember
- User's password at any given site is hard to guess
- Compromise of password at one site does not permit immediate compromise at other sites

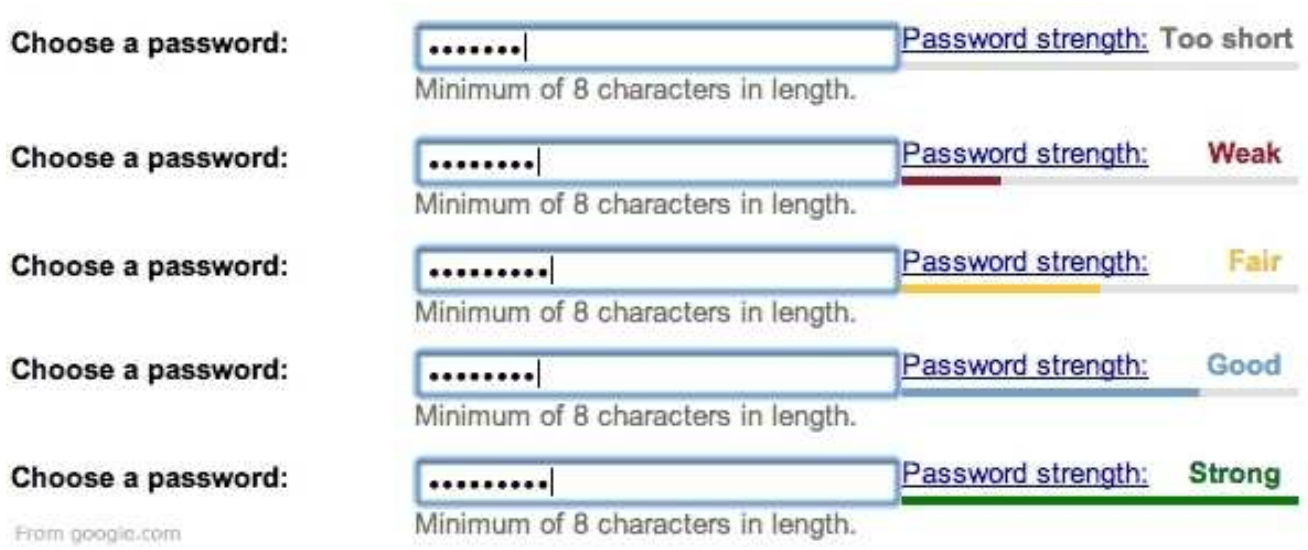
But:

- Must still **protect** and **remember** strong **master password**

Password Strength Meter

Gives user feedback on the strength of the password

- Intended to **measure guessability**
- Research shows that these **can work**, but the design must be **stringent** (e.g., forcing unusual characters)
  - Ur et al, "How does your password measure up? The effect of strength meters on password creation", Proc. USENIX Security Symposium, 2012.



Better together

Password manager

- **One security decision**, not many

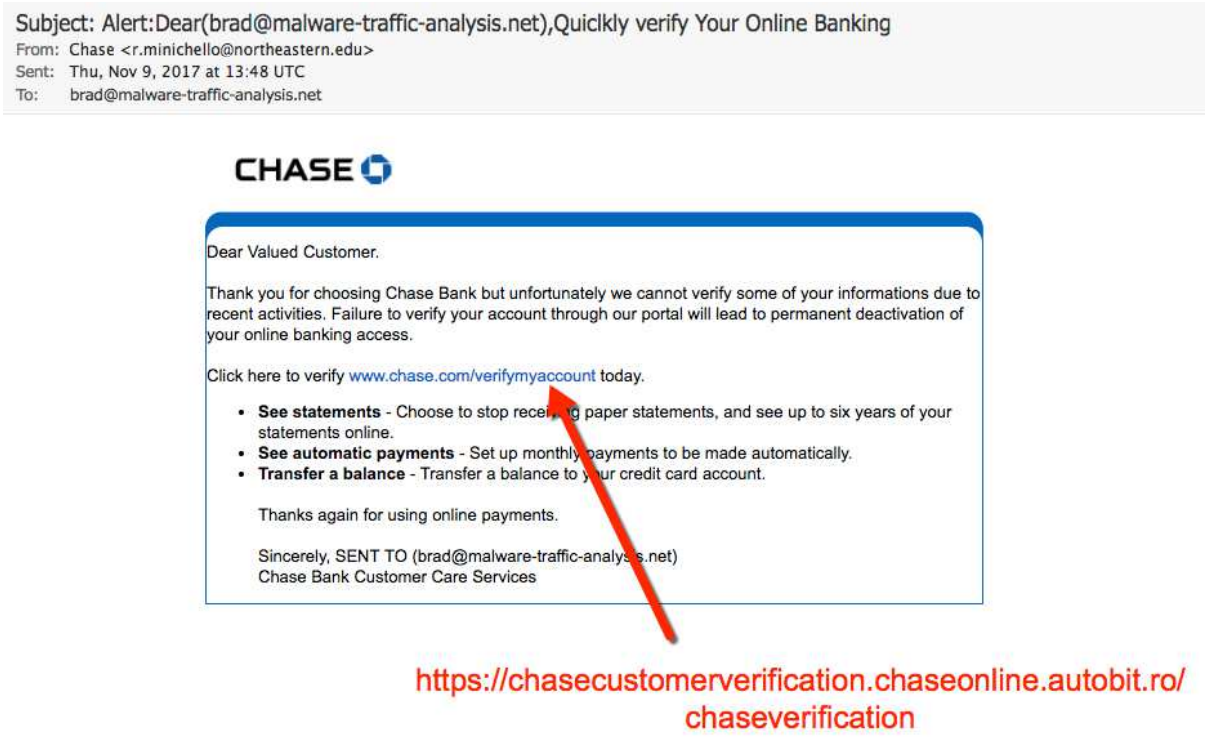
Password meter

- Users can **explore ramifications of various choices** by visualizing quality and reasoning of password
- **Do not permit poor choices** (or reduce the chances of them) by enforcing a minimum score

Phishing

User is tricked into thinking that a **site** or **e-mail** **is legitimate**, rather than a scam

- And is then tricked into installing malware or performing other harmful actions



Phishing

Failure: Site or e-mail not (really) authenticated

Internet e-mail and web protocols **not originally designed for remote authentication**

Solution is **hard to deploy**

- Use hard-to-fake notions of identity, like **public key cryptography**
  - But which system?
  - How to upgrade gradually?

Design Category:
Trust with Reluctance

Trust with Reluctance (TwR)

Whole system security depends on the **secure operation of its parts**

- These parts are **trusted**

So: **Improve security by reducing the need trust**

- By using a better design
- By using a better implementation process. For example by using a type safe language.
- By not making unnecessary assumptions
  - If you use third party code, how do you know what it does?
  - If you are not a crypto expert, why do you think you can design/implement your own crypto algorithm?

Categories: Prevention and mitigation

TwR: Small Trusted Computing Base (TCB)

Keep the **TCB small** (and simple) to **reduce overall susceptibility to compromise**

- TCB comprises the system components that *must* work correctly to ensure security
- **Category**: Prevention

Example: **Operating system kernels**

- Kernels enforce security policies, but are often millions of lines of code
  - Compromise in a device driver compromises security overall
- Better: Minimize size of kernel to reduce trusted components
  - Device drivers moved outside of kernel in micro-kernel designs

Failure: Large Trusted Computing Base (TCB)

Security software is part of the TCB

But as it grows in size and complexity, it becomes vulnerable itself, and can be bypassed

- Over time, this software has grown in size and complexity and is now has become vulnerable itself and is frequently attacked.
- Zero-day bugs in **Kaspersky** and **FireEye** products found, exploits disclosed
  - https://www.helpnetsecurity.com/2015/09/08/zero-day-bugs-in-kaspersky-and-fireeye-products-found-exploits-disclosed/
- Google's Project Zero reveals update flaws in Malwarebytes' antivirus software
  - https://www.v3.co.uk/v3-uk/news/2444776/googles-project-zero-reveals-update-flaws-in-malwarebytes-antivirus-software

TwR: Least Privilege

Don’t give a part of the system more privileges than it needs to do its job (“*need to know*”)

- **Category**: Mitigation

Example: Attenuate delegations

- Mail program delegates to editor for authoring mails
  - `– vi, emacs`
- But many editors permit escaping to a command shell to run arbitrary programs: too much privilege!
- Better Design: Use a restricted editor (`pico`)

Lesson: Trust is Transitive

If you trust something, you trust what it trusts

- *This trust can be misplaced*

Previous e-mail client example

- Mailer delegates to an arbitrary editor
- The editor permits running arbitrary code
- Hence the mailer permits running arbitrary code

Rule: Input validation

Input validation is a **kind of least privilege!**

- You are **trusting a subsystem** only **under certain circumstances**
- *Validate that those circumstances hold*

Several **examples** so far:

- Trust a given function *if* the range of its parameters is limited (e.g., within the length of a buffer)
- Trust a client form field *if* it contains no <script> tags (and other code-interpretable strings)

TwR: Promote Privacy

A good overall system goal is to **restrict flow of sensitive data** as much as possible

- Doing so promotes privacy by *reducing trust/privilege*
- **Category**: Mitigation

Example: A student admission system receives (sensitive) letters of recommendation as PDF files

- A typical design would allow reviewers to download these files for viewing on their local computers
  - But then compromise of these computers leaks private information

Better: PDFs only viewable in browser; no data downloaded to client machine.

TwR: Compartmentalization

Isolate a system component in a compartment, or **sandbox**, reducing its privilege by making certain interactions impossible

- **Category**: Prevention and Mitigation

Example: Disconnect student records database from the Internet

- Grant access only be direct terminals

Example: `Seccomp` system call in Linux

- Enables compartments for untrusted code

SecComp

Linux system call enabled since 2.6.12 (2005)

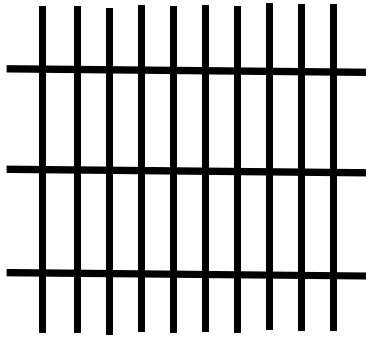
- Affected process can subsequently **only perform `read`, `write`, `exit`, and `sigreturn` system calls**
  - No support for open call: Can only use already-open file descriptors
- **Isolates a process by limiting possible interactions**

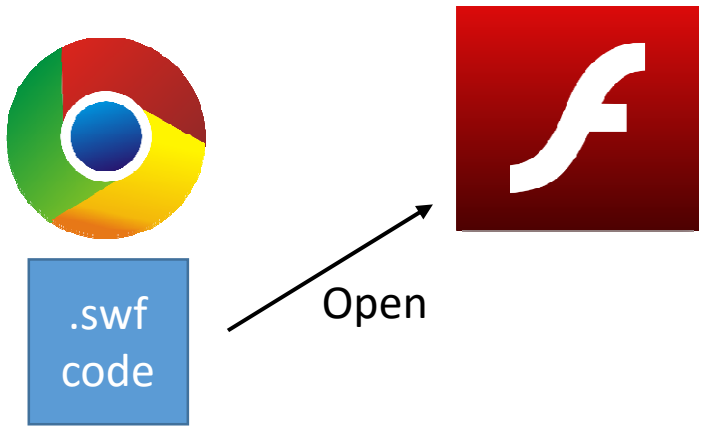
Follow-on work produced **`seccomp-bpf`**

- **Limit process to policy-specific set of system calls**, subject to a policy handled by the kernel
  - Policy akin to *Berkeley Packet Filters (BPF)*
- Used by *Chrome*, *OpenSSH*, *vsftpd*, and others

dea: Isolate Flash Player

- Receive `.swd code`, save it
- Call `fork` to create a new process
- In the new process, open the file
- Call `exec` to run Flash player
- Call `seccomp-bpf` to compartmentalize





Design Categories: Defense in Depth Monitoring/Traceability

Defense in Depth (DiD)

Security by diversity

- If one layer is broken, there is another of a materially different character that needs to be bypassed
- **Categories**: Prevention/Mitigation

Example: Do *all of the following*, not just one

- Use a firewall for preventing access via non-web ports
- Encrypt account data at rest
- Use a safe language for avoiding low-level vulnerabilities

Failure: Authentication Bypass

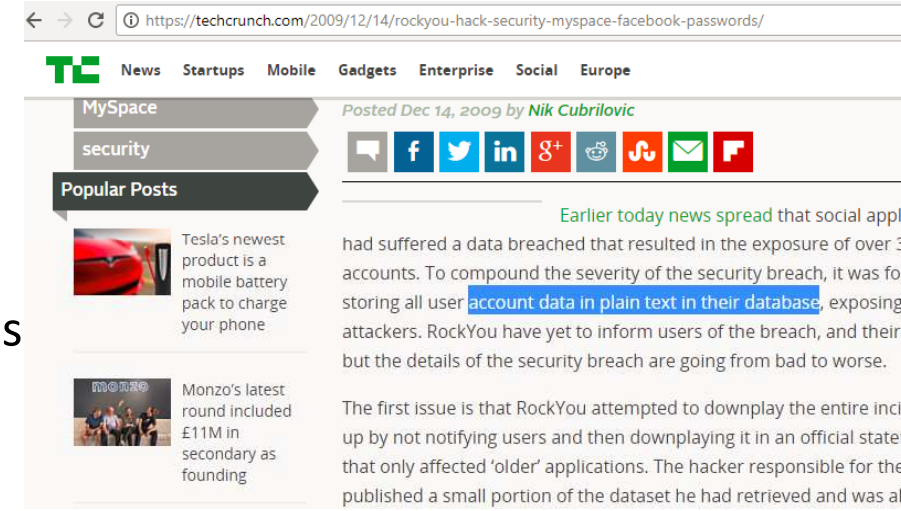
(Poor) passwords can be guessed

- bypassing authentication process intent

Passwords can be stolen

- **Defense in depth**: Should encrypt the password database
  - Assumes that compromise is possible, and thus requires additional defense

RockYou was compromised and their password database was stolen. All of the passwords were stored in plain text exposing all of that information to attackers.



https://techcrunch.com/2009/12/14/rock
hack-security-myspace-facebook-passwo

DiD: Use community resources

Use hardened code, perhaps from other projects

- E.g., **crypto libraries**
- But make sure it meets your needs (*test it*; cf. Heartbleed!)

Vet designs publicly: *No security by obscurity*

Stay up on recent threats and research

- **NIST** for **standards**
- **OWASP**, **CERT**, **Bugtraq** for **vulnerability reports**
- **SANS** *Newsbites* for **latest top threats**
- Academic and industry **conferences and journals** for **longer term trends**, **technology**, and **risks**

Failure: Broken Crypto Implementation

Getting crypto right is hard many things that you could get wrong.

- **Timing channel against RSA**
  - Remote timing attacks are practical, Usenix 2003
  - Remote timing attacks are still practical, ESORICS 2011
- **Poor randomness**
  - Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices, USENIX 2012
  - Weak Keys Remain Widespread in Network Devices, IMC '16

Use vetted implementations and algorithms and be sure that you are using them correctly.

Monitoring and Traceability

If you are attacked, how will you know it?

- Once you learn, **how will you discern the cause?**

Software must be designed to **log relevant operational information**

- What to log? E.g., events handled, packets processed, requests satisfied, …
- **Category:** Detection and Recovery

Log aggregation: Correlate activities of multiple applications when diagnosing a breach

- E.g., `splunk` log aggregator https://www.splunk.com/